COMB: A Portable Benchmark Suite for Assessing MPI Overlap

William Lawry, Christopher Wilson, Arthur B. Maccabe [*]
University of New Mexico, Computer Science Department
Farris Engineering Center, Rm 157, Albuquerque NM 87131
{bill,riley,maccabe}@cs.unm.edu

Ron Brightwell [†]
Sandia National Laboratories
Scalable Computing Systems Department, 9223
P. O. Box 5800, Albuquerque, NM 87185-1110 bright@cs.sandia.gov

Abstract

This paper describes a portable benchmark suite that assesses the ability of cluster networking hardware and software to overlap MPI communication and computation. The Communication Offload MPI-based Benchmark, or COMB, uses two methods to characterize the ability of messages to make progress concurrently with computational processing on the host processor(s). COMB measures the relationship between MPI communication bandwidth and host CPU availability.

1. Introduction

Advances in networking technology for cluster computing have led to significant improvements in achievable latency and bandwidth performance. Many of these improvements are based on an implementation strategy called Operating System Bypass, or OS-bypass, which attempts to increase network performance and reduce host CPU overhead by offloading communication operations to intelligent network interfaces. These interfaces, such as Myrinet [3], are capable of "user-level" networking, that is, moving data directly from an application's address space without any involvement of the operating system in the data transfer.

Overhead has been shown to be the most significant factor in effecting application performance [4]. Unfortunately, the reduction in host CPU overhead has not always been accompanied by the performance benefits afforded by overlap of communication and computation. Indeed, White and Boval [1] concluded that the MPI implementations commonly targeted for large-scale parallel computation do not show significant performance improvement when MPI programs are designed for overlap, based on experimental models.

While most MPI microbenchmarks can measure latency, bandwidth, and host CPU overhead, they fail to accurately characterize the actual performance that applications can expect. Communication microbenchmarks typically focus on message passing performance relative to achieving peak performance of the network and do not characterize the performance impact of message passing relative to *both* the peak performance of the network and the peak availability of the host CPU to the application. For a given message size, the COMB benchmark indicates the maximum possible sustained bandwidth as well as the CPU availability to the application at this level of maximum communication.

The benchmark provides an additional check of whether the MPI library complies with the Progress Rule of the MPI Standard: that non-local message passing operations will complete independently of a process making library calls. This check is based on one of several MPI-call durations provided by COMB.

2. COMB communication models

COMB uses two communication models. Using communication between a pair of nodes, the two models jointly target two key aspects of cluster communication, overlap and overhead. While two-node communication provides a useful but limited view to the purchaser or user of a communication system, it can give the developer meaningful information into the problems and bottlenecks in the current state of the implementation of interest. A developer can use

[*]This work was supported in part through the Computer Science Research Institute (CSRI) at Sandia National Laboratories under contract number SF-6432-CR.

[†]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

accurate and detailed metrics for realizing specific performance enhancements which contribute to the ultimate goal of comprehensive performance.

COMB avoids the non-portable and/or less than accurate methods of measuring overhead such as a counter in the idle loop of the kernel or in a separate user-level thread, or such as the use of system contextual information. Kernel counters are very accurate but not portable. Separate user-level threads include overhead due to context switching. System-based contextual information, like **pstat**, is system dependent and can have a granularity dependent on the the OS scheduler.

COMB avoids these issues by using one process on each of two nodes. One process, the *support* process, simply supports communication while the other process, the *timing* process, communicates as well as monitors its rate of progress in **for-loop** iterations. This two-node communication is employed in two distinct methods: *Polling* and *Post-Work-Wait* as illustrated in Figure 1. Both methods actually run in two phases. During the first phase, the *dry work* phase, the method records the amount of time to accomplish a predetermined amount of work in the absence of communication. The second *wet work* phase records the time for the same amount of work while the two processes are exchanging messages. The CPU availability is reported as:

$$\text{availability} = \frac{\text{time(work without messaging)}}{\text{time(work plus MPI calls while messaging)}}$$

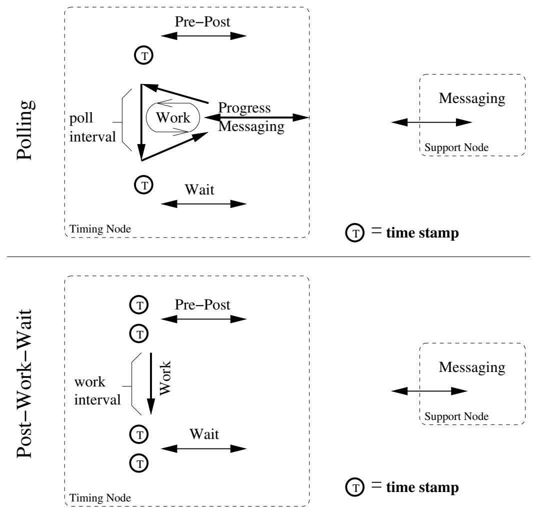

Figure 1. COMB methods: Polling & Post-Work-Wait

The *Polling* method periodically polls with **MPI_Test** for message arrival. After some number of iterations, the *timing* process polls for message arrival; this number of iterations defines the polling interval. If a test for message completion is negative, the *timing* process will iterate through another polling interval before testing again. If a test for completion is positive, the process will post related messaging calls and will similarly address any other received messages before entering another polling interval. The support process sends messages as fast as they are consumed. The *Polling* method terminates when a pre-determined number of accumulated iterations completes.

The essentially artificial communication model of the *Polling* method is aimed at gaging the maximum possible availability of the CPU as experienced by a user-level application under a continuous communication environment. In contrast, the *Post-Work-Wait* method models the exchange of two messages between nodes where only the essential non-blocking calls are made before some computational work (i.e., **for-loop** iterations) which is finally followed by a wait for message completion. The number of "work" iterations may be varied for a view into how the communication-specific time is spent during the non-blocking post, work, or wait periods. Again, the method is illustrated in Figure 1.

3. Platform description

The results presented here are based on the same node and network hardware using a variety of communication software. Each node contained a 500 MHz Intel Pentium III processor with 256MB of main memory and a Myrinet [3] LANai 7.2 network interface card (NIC). Nodes were connected using a Myrinet 8-port SAN/LAN switch. As discussed in detail below, we use two types of communication systems: MPICH/GM and MPI/Portals. These systems represent different communication paradigms and herein serve to illustrate the system characterization afforded by COMB rather than serving as points supporting a position on which paradigm is better.

MPICH/GM GM [2] is the supported message passing software from Myricom for Myrinet. It consists of a user-level library, a Linux driver and Myrinet Control Program (MCP) which runs on the NIC. Myricom also supplies a port of the MPICH [8] implementation of the MPI Standard. Our results were gathered using GM version 1.4, MPICH/GM version 1.2..4, and a Linux 2.2.14 kernel. This setup makes use of the intelligent NIC such that payload transfer to and from user-space occurs with little host involvement.

MPI/Portals Results were also gathered using the Portals 3.0 [7, 6] software designed and developed by Sandia and the University of New Mexico. Portals is an interface for data movement designed to support massively parallel commodity clusters, such as the Computational Plant [5]. We have also ported the MPICH implementation of MPI to Portals 3.0.

The particular implementation of Portals for Myrinet used in our experiments is kernel-based. The user-level Portals library interfaces to a Linux kernel module that processes Portals messages. This kernel module in turn interfaces to another kernel module that provides reliability and flow control for Myrinet packets. This kernel module works with a Sandia-developed MCP that simply acts as a packet engine. This particular implementation of Portals does not employ OS-bypass techniques.

4. Results and analysis

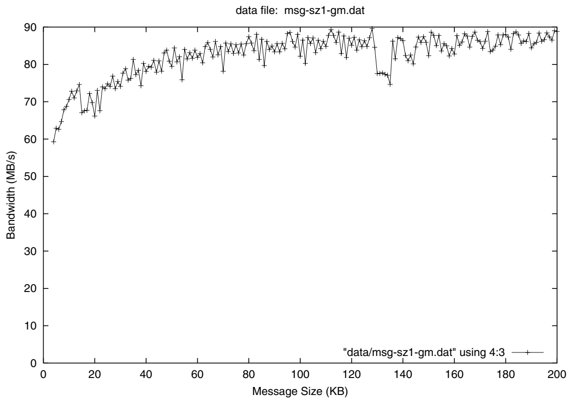

Figure 2. Polling: bandwidth per message size (MPICH/GM)

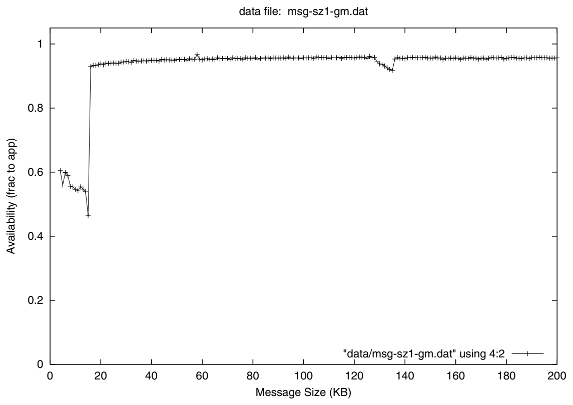

Figure 3. Polling: availability per message size (MPICH/GM)

Figures 2 and 3 show bandwidth and availability results, or equivalently overhead results, for using MPICH/GM using a short poll interval which ensures a maximum possible messaging rate. The pair of data points for a particular message size in these two graphs result from the same timing run; the experienced availability, or equivalently overhead, is concurrently measured with network bandwidth. For these two graphs, the polling interval is sufficiently small in each run such that the bandwidth graph represents the maximum sustainable bandwidth for that message size. Note that the graphs bring out system characteristics with respect to MPICH/GM's protocol change at 16 KB, to another system pattern that repeats every 16 KB between message sizes between 16 KB and 128 KB, as well as to a second MPICH/GM protocol change at 128 KB. The run-time for gathering this data is approximately five minutes on our hardware platform.

4.1. Communication and computation overlap

When taken together, Figures 2 and 3 clearly illustrate the potential in MPICH/GM to overlap communication and computation. This represents MPICH/GM's actual overlap under *Polling's* artificially frequent calls to **MPI_Test** and the representation is made in terms of actual CPU availability while communication is in progress. Alternatively and as previously discussed, the *Post-Work-Wait* method model is closer to a realistic communication pattern between two nodes; during the exchange of two messages, the *timing* process performs work after the set of non-blocking calls and before the wait for message completion. A similar experimental technique has been used elsewhere [1].

Since *Post-Work-Wait* times the durations of the three periods, consider the duration of the wait period as a function of increasing work interval. If a communication system can overlap communication and computation with only the initiating posts then the wait duration should decrease with a longer and longer work interval. Figure 4 shows this function for the MPICH/GM and MPI/Portals systems. Clearly, without additional calls to the MPI library, MPICH/GM does not have overlap whereas MPI/Portals does.

Again, note that this is not an overall system characterization but a specific qualification of the ability of a system to overlap communication. In other comparisons, MPICH/GM has comparatively good performance in terms of raw bandwidth and general lack of communication overhead and subsection 4.2 illustrates the communication overhead associated with MPI/Portals.

4.2. Communication overhead

Figure 5 shows the *Post-Work-Wait* timing signature for MPI/Portals with 100 KB messages. The "y" axis is used to indicate the durations for the three periods of the method and the "x" axis indicates the increasing interval between

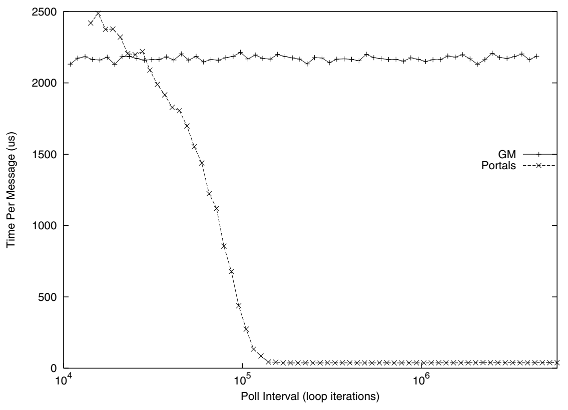

Figure 4. PWW: durations for MPI's wait

the non-blocking posts and the wait for message completion. As previously discussed, the wait durations decline with increasing work interval showing that MPI/Portals is able to overlap communication and computation.

What is also shown is the effect of overhead on delaying computation. Such overhead is indicated by the difference between the work without message handling ("work") and work with message handling ("work (MH)"). Also, the overhead associated with the non-blocking posts is evident as the difference between a) the work with message handling and b) the sum of the non-blocking posts and work with message handling ("post work"). This detailed representation of the effects of communication are an aid to the developer in gaging efforts at performance improvement for specific periods within a communication cycle.

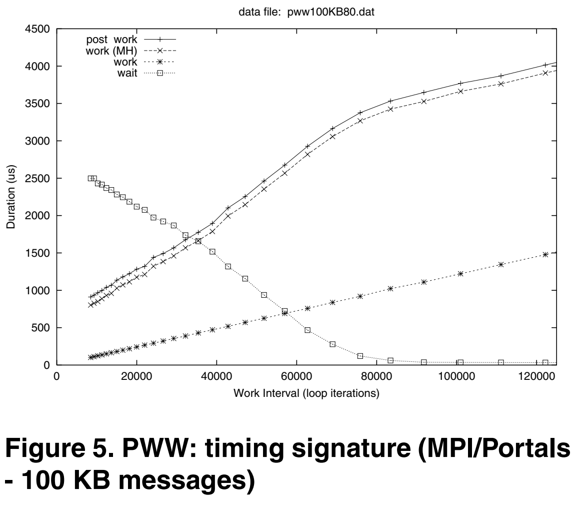

Figure 5. PWW: timing signature (MPI/Portals - 100 KB messages)

5. Summary

In this paper, we have described the COMB benchmark suite that characterizes a systems communication overhead as well as the ability of a system to overlap computation and communication. We have described the methods and approach of COMB and demonstrated its utility in providing insight into the underlying implementation of communication system. In particular, we have demonstrated the benchmark suite's ability to distinguish between systems that support the MPI Progress Rule and those that do not. COMB is available at `http://www.cs.unm.edu/~maccabe`.

References

[1] James B. White and Steve W. Bova. Where's the overlap? overlapping communication and computation in several popular mpi implementations. In *Proceedings of the Third MPI Developers' and Users' Conference*, Mar. 1999.

[2] Myricom, Inc. The GM message passing system. Technical report, 1997.

[3] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet-a gigabit-per-second local-area network. In *IEEE Micro*, volume 15(1), pages 29–36, Feb. 1995.

[4] Richard P. Martin, Amin M. Vahdat, David E. Cullet, and Thomas E. Anderson. Effects of communication latency, overhead, and bandwidth, in a cluster architecture. In *Proceedings of the 24th Annual International Symposium on Computer Architecture(ISCA-97), Computer Architecture News*, volume 25,2.

[5] Ron B. Brightwell, Lee Ann Fisk, David S. Greenberg, Tramm B. Hudson, Michael J. Levenhagen, Aruthur B. Maccabe, and Rolf Riesen. Massively parallel computing using commodity components. In Parallel Computing, volume 26, pages 243–266, Feb. 2000.

[6] Ron Brightwell, Bill Lawry, Arthur B. Maccabe, and Rolf Reissen. Portals 3.0: Protocol building blocks for low overhead communication. In CAC Workshop, Apr. 2002.

[7] Ron Brightwell, Tramm Hudson, Rolf Riesen, and Arther B. Maccabe. The Portals 3.0 message passing interface. Technical report SAND99-2959, Sandia National Laboratories, Dec. 1999.

[8] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. In *Parallel Computing*, volume 22(6), pages 789–828, Sept. 1996.